UCRL-ID-151859

On the Performance of Parallel Algebraic Multigrid

Arne Nagel, Robert Falgout

February 18, 2003

DISCLAIMER

This work was performed under the auspices of the U. S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under Contract No. W-7405-Eng-48.

On the Performance of Parallel Algebraic Multigrid

Arne Nägel*

October 2002

Abstract

As algebraic multigrid (AMG) can be applied to a wide variety of problems on extremely large, unstructured grids on the one hand and massively parallel computer systems are available on the other hand, there has been a need for parallel implementations of AMG. This report analyses the performance of BoomerAMG[1], a parallel algorithm developed at Lawrence Livermore National Laboratory.

Abstracting the implementation we describe the basic components of the costly setup phase and analyse their behavior in a (massively) parallel distributed memory environment. We present numerical results, compare them to the developed theory and finally aim at possible improvements in the future.

*anaegel@ix.urz.uni-heidelberg.de

1 Introduction

After some remarks on general assumptions and notation, we are going to analyze the three major components of the AMG setup phase. Sections 2 and 3 deal with details of the coarse grid selection, sections 4 and 5 analyse the computation of interpolation and the coarse grid operator, while section 6 finally contains test results.

1.1 Parallelization model

Target platforms for this work are in general, parallel distributed memory architectures. We assume that message passing strategies are implemented in MPI and, for reasons of simplification we do not exploit the fact that some target platforms may support shared memory within clusters. As we are interested in solving linear systems of the form

$$\begin{aligned} Ax &= b, \\ A = (a_{ij}), x = (x_i), b &= (b_i) \end{aligned}$$

arising from partial differential equations, the equations and their data are distributed naturally using domain-partitioning. We make the following simplifying assumptions:

Let Ω be the original grid, let Π be the set of processors involved and let each processor hold a subset Ω_p of grid points. We assume maximal symmetry, i.e. let each Ω_p be a cube of size N. We denote the ghost points, i.e. all off-processor neighbors of nodes on processor p, by $\partial\Omega_p = \{j \in \Omega | \exists i \in \Omega_p : a_{ij} \neq 0, \pi(j) \neq p\}$. According to our assumptions about the grid structure, the cardinality of this set depends on d, the degree of the graph. We assume that the maximal distance between a point in $\partial\Omega$ and the closest point in Ω is given by

$$r = \left\lceil \frac{\sqrt[3]{d} - 1}{2} \right\rceil \tag{1.1}$$

in 3-D, which represents the number of neighbors of the center point in a cube of size d. To estimate the cardinality of $\partial\Omega$, we use the formula

$$|\partial\Omega_p| = 3\sqrt[3]{N}^2 2r + 3\sqrt[3]{N}(2r)^2 + (2r)^3. \tag{1.2}$$

Analogies in two dimensions may be developed straight forward. In the remainder of this work, the relation

$$|\partial\Omega_p| = \sum_{q \in N_p} |\partial\Omega_q \cap \Omega_p| \tag{1.3}$$

will be important, when A is symmetric.

1.2 Platform specific details

The most important aspect in this analysis will of course be the trade-off between computation and communication. To quantify these sizes and to relate them to the actual time needed, we use the
following concept. We define:

The time (in seconds) for passing a message of m doubles from one processor to another is given by

$$T_{comm}(m) = \alpha + \beta m. \tag{1.4}$$

In some passages the variable $\nu = \nu(N, d, \Omega)$ is used to bound the number of neighbors of an arbitrary processor.

The time (in seconds) for n floating point operations (flops) on a fixed processor is given by

$$T_{comp}(n) = \gamma n. \tag{1.5}$$

Representative values for IBM ASCI Blue are

$$\begin{aligned} \alpha &= 59\,\mu sec, \\ \beta &= .25\,\mu\text{sec / double}, \\ \text{and}\,\gamma &= .06\,\mu\text{sec / flop} \end{aligned}$$

1.3 Compressed Row Storage

The Compressed Row Storage (CRS) format offers the possibility to store a sparse matrix $A = (a_{ij})$ without further assumptions on the sparsity structure. It stores the nonzeroes of the matrix in three contiguous memory locations using an indirect addressing scheme explained in the remainder of this subsection. Note that this fact influences, for example, the efficiency of matrix/vector products. For a quick overview and additional information on different storage formats refer to [5].

Let us assume in this subsection, that A has n rows and nz nonzero entries. To store the matrix, three different vectors are initialized: one of size $n + 1$, denoted by a_i, and two of size nz, denoted by a_j and a_{data}, respectively. The actual nonzero entries are stored in the latter one, row by row starting in the upper left hand corner, whereas the other ones are used for a consistent indirect addressing strategy. Each entry in a_i represents an offset for a single row of A in the two other vectors, e.g. if we are interested in the k-th row of A, we have to inspect all slots s satisfying $a_i[k] \leq s < a_i[k + 1]$. The entries in a_j indicate,

which column the data in a_{data} belongs to.

The scheme should be comprehensible after looking at the example:

$$A = \begin{pmatrix} 1 & 2 & 0 & 0 \\ 3 & 4 & 5 & 0 \\ 0 & 6 & 7 & 8 \\ 0 & 9 & 0 & 10 \end{pmatrix},$$

which will be stored in the vectors

a_i :

1	3	6	9	11

a_j :

1	2	1	2	...	2	4

a_{data} :

1	2	3	4	...	9	10

.

2 Coarse Grid Selection

The selection of the coarse grid is the only one of the three analysed components, which is explicitly non-local by nature. Typical of AMG all coarsening schemes are derived using concepts of dependence and influence. We say that point i depends on point j, if the value of the unknown j is important in determining the value i. Thus we denote by S_i the set of points point i depends on and led by M-matrix properties we define in analogy to the original approach by Ruge and Stüben:

$$S_i = \{j \neq i : -a_{ij} \geq \theta max(-a_{ik})\}. \tag{2.1}$$

Note that S_i may be identified with a row of a matrix $S = (s_{ij})$. The set of points depending on point i is thus denoted by S_i^T. Additionally we define the coarse neighborhood (of point i) $C_i = S_i \cap C$. We are now giving a brief overview of the classical (sequential) algorithm and look at how it may be employed in BoomerAMG.

2.1 Ruge-Stüben Coarsening

The classical Ruge-Stüben coarsening (RS) is based on the criteria:

(C1) For each $i \in F$, each $j \in S_i$ is either in C or in $S_j \cap C_i$.

(C2) C should be a maximal subset with the property, that no point in C depends on another point in C.

We implement these heuristics using the measure

$$\mu(i) = |S_i^T|, \forall i \in \Omega, \tag{2.2}$$

which is iteratively used to find a point i most valuable to be part of the coarse grid. As, according to the heuristics, its neighborhood is affected by this decision, measures in the neighborhood are changed afterwards. The value μ will be used as a key in the hashtable U and we use the pseudo code:

$C \leftarrow \varnothing$;
$F \leftarrow \varnothing$;
while $U \neq \varnothing$ {
    select $i \in U[m_{max}]$;
    $C \leftarrow C \cup \{i\}$;
    $U[m_{max}] \leftarrow U[m_{max}] \setminus \{i\}$;
    for each $j \in S_i^T$ {
        $m \leftarrow \mu(j)$;
        $F \leftarrow F \cup \{j\}$;
        $U[m] \leftarrow U[m] \setminus \{j\}$;
        for each $k \in S_j$ {
            $m \leftarrow \mu(k)$;

```
                U[m] ← U[m] \ {k};
                U[m + 1] ← U[m + 1] ∪ {k};
            }
        }
        for each j ∈ S_i {
            m ← μ(j);
            U[m] ← U[m] \ {j};
            U[m − 1] ← U[m − 1] ∪ {j};
        }
}
```

In order to enforce **(C1)** rigorously, we perform a second pass removing F-F dependencies.

```
for each i ∈ F {
    for each j ∈ C_i {
        mark j;
    }
    for each j ∈ F_i {
        for each k ∈ S_j {
            check, if k is marked;
        }
    }
    for each j ∈ C_i {
        unmark j;
    }
}
```

The algorithm has a straight forward structure and runs in $\gamma O(Nd^2)$, if we assume O(1) cost for removing and adding entries to the lists $U[m]$. (Note that each point in the neighborhood of a selected point is very likely to have lots of connections too.)

2.2 Coarsening using CLJP

Based on a graph coloring algorithm by Jones and Plassmann presented in [3], CLJP [4] is a slight variation based on modified measures $\mu(i) = |S_i^T| + \rho(i)$, i.e. the number of points influenced by i incremented by a random number in (0,1), which is used to break ties. The heuristics applied are slightly different from the previous ones:

(H1) Values at c-points are not interpolated; hence, neighbors that influence a c-point are less valuable as potential c-points themselves.

(H2) If k and j both depend on a given c-point i, and j influences k, then j is less valuable as a potential c-point, k can be interpolated from i.

Although details about the implementation are postponed to the following section, remark the similarity of the algorithm to the RS-approach:

$C \leftarrow \varnothing$;
$F \leftarrow \varnothing$;
for each $i \in V\{$
    $\mu(i) = |S_i^T| + \rho(i)$;
}
while $V \neq \varnothing$ {
        select an independent set $D = \{i|\mu(i) > \mu(k) \forall k \in S_i \cup S_i^T\}$;
        $C \leftarrow C \cup D$;
        change measures, i.e. apply heuristics to $V \leftarrow V \setminus D$;
        select further f-points according to changed measures;
}

2.3 Falgout Coarsening

Tests in [1] show, that a hybrid RS/CLJP scheme called Falgout Coarsening, achieves very satisfying results. As the RS scheme produces sparse coarse grids and reaches some near optimal results in many geometric cases, it seems to be a promising approach when applied to the interior of the processor. Unfortunately, the boundaries have to be treated in a different way to avoid dependecies between F points, violating (C1), which may lead to a packing of c-points in this area. As the CLJP algorithm explained below is based on similar heuristics, but introduces a random component to categorize points, it is very likely to produce a larger number of C-C connections in the interior of the processor than desirable. Falgout coarsening avoids this by choosing a set of coarse points according to the RS scheme and using this data (while neglecting decisions made on the boundary) as first independent set for CLJP. Once the CLJP heuristics are applied, all points in the interior become f-points and CLJP operates on the boundary only in the following steps.

According to the CLJP reduction model derived, it is desirable to start off with the smallest possible degrees locally. An additional feature of the model is that the algorithm seems to reduce graphs with strongly varying degrees very efficiently, which would make this hybrid scheme a very effective algorithm for treating the boundaries. Unfortunately, the growing stencil increases the number of points in this area too.

3 Cost of CLJP

The implementation of the CLJP algorithm in BoomerAMG basically consist of one loop marking all points as C or F points which aborts when all points (or vertices according to graph theory) have been colored. The computational side of the loop can be performed locally, but in between some communication is required to ensure consistent coloring on the processor boundaries. Although pseudo code is given in a general version following [2], we assume in the analysis itself, that A is symmetric. No matter, the formulas derived should also work well for a non-symmetric A, if we have symmetry over the the processor boundaries in the sense, that $|\Omega_p \cap \partial\Omega_q| = |\Omega_q \cap \partial\Omega_p|$, $\forall p, q \in \Pi$. This is obviously true in the symmetric case, as the sets are identical.

3.1 The setup phase

Before the actual computation may start, we have to make sure that each processor is able to compute the weights correctly and is informed about strength information in its ghost area. As this is information stored in the column of the strength matrix S and we assume CRS storage format, we have to submit row information related to nodes on different processors' $\overline{\Omega}_q$ in advance. Receiving the symmetric data, we define the environment of points sending information for the later evaluation of local maxima and collect the missing information of S_i^T, $\forall i \in \Omega_p$ After this, we compute the measures $\mu_p(i)$, the contribution of processor p for weights in each point $i \in \overline{\Omega}_p$.

send filtered rows S_i, i.e. $S_i^\delta = \{s_{ij} | i \in \Omega_p, \exists q : j \in \overline{\Omega}_q\}$;
receive filtered rows S_i^δ;
compute $\mu_p(i)$, $\forall i \in \overline{\Omega}_p$;

Doing all this is rather cheap, using the assumption about symmetry the costs are bounded by

$$T_{setup} = \nu\alpha + \beta|\partial\Omega_p|d + \gamma|\overline{\Omega}_p|d \tag{3.1}$$

(Note: In the actual implementation the sets $S_{ext} = \bigcup S_i^\delta$ are computed by determining necessary entries for the matrix multiplication SA.)

3.2 The main loop

The original algorithm is based on the selection of a maximum independent set D according to the measures, an iterative extension of C using D and the application of the heuristics according to the chosen C points. In the current code this is implemented in a two pass strategy, both of the passes require additional communication.

In the k-th step the graph $G_k = (V_k, E_k)$ is defined by the undetermined vertices V_k and remaining edges E_k and we start with an initial graph $G_0 = (V_0, E_0)$. If $k \neq 0$ let us assume, that in the previous steps sets of coarse and

fine grid points, C_j and F_j, were removed for all $j < k$. The degree of the graph G_k is denoted by d_k; to simplify notation we use the same notation for the local degree, i.e. the degree in each grid point. Furthermore we identify sets and their cardinal numbers.

The **first pass** determines local maxima and thus requires the most recent information about the weights, which are determined/changed by the setup hase/the second pass. The necessary update consists of sending ghost measures

changes (and receiving values for non-ghost points), a computational update and sending the updated values to all neighbors.

send ghost measures decrements, $\forall i \in \partial\Omega_p$;
receive measure decrements, $\forall i \in (\bigcup_{q \neq p} \partial\Omega_q) \cap \Omega_p$;
update measures, i.e. subtract decrements, $\forall i \in \Omega_p$;
send measures $\mu(i)$, $\forall i \in (\bigcup_{q \neq p} \partial\Omega_q) \cap \Omega_p$;
receive ghost measures $\mu(i)$, $\forall i \in \partial\Omega_p$;

After this, the values $\mu(i)$ are known locally for every $i \in \overline{\Omega}_p$ and every $p \in \Pi$. The update costs are given by

$$T_{comm} = 2\nu\alpha + 2\beta \sum_{q \in N_p} |\partial\Omega_p \cap \Omega_q| + \gamma \sum_{q \in N_p} |\partial\Omega_q \cap \Omega_p| \tag{3.2}$$

Now the identification of local maxima and some operations on the previously found coarse and fine grid points are performed. Note, that we might claim for some points on the boundary to maximize μ in the environment on processor p, which are not local maxima repects to other processors.

for each $i \in G_k$ {
  if $i \in C_k \cup F_k$
    remove i from V_k;
  }
  else {
    for each $j \in S_i$
      if $\mu(i) > \mu(j)$ then $j \notin C_{k+1} \cup C_{k+1}^{\delta}$;
      if $\mu(j) > \mu(i)$ then $i \notin C_{k+1}$;
    }
  }
}
for each $i \in G_k^{\delta}$ {
  if $i \in C_k^{\delta} \cup F_k^{\delta}$
    remove i from V_k^{δ};
  }
  else {
    for each $j \in S_i^{\delta}$
      if $\mu(i) > \mu(j)$ then $j \notin C_{k+1} \cup C_{k+1}^{\delta}$;

if $\mu(j) > \mu(i)$ then $i \notin C^{\delta}_{k+1}$;
}
}
}

We divscover that the overall costs to do this are (note the dramatic size of the second addend for large d):

$$T_{1st} = \gamma(V_k + (V_k - C_k - F_k)d) + \gamma(V_k^{\delta} + (V_k^{\delta} - C_k^{\delta} - F_k^{\delta})d^{\delta}) \tag{3.3}$$

In a global exchange step, the global graph size is computed, to determine, if the algorithm already converged. The costs are

$$T_{sync} = (\alpha + \beta)2log(P) \tag{3.4}$$

A true local maximum is recognized as such on all involved processors:

send ghost cf-values, $\forall i \in \partial\Omega_p$;
receive cf-values, $\forall i \in \Omega_p, \pi(i) \neq p$;
compare cf-values, $\forall i \in \Omega_p, \pi(i) \neq p$, correct falsely identified local maxima;
send cf-values$\mu(i), \forall i \in \partial\Omega_p$;
receive ghost cf-values, $\forall i \in \partial\Omega_p, \pi(i) \neq p$;

Costs are the same as for the first communication step; the comparison should not be too expensive, it may i.e. be implemented by putting the maximum of two values into the local cf-value vector.

Last, but not least, the **second pass** is performed, i.e. the heuristic are applied:

for each $i \in V_k$
if $i \in C_k$ {
for each $j \in S_i$ {
mark (i,j) for removal;
decrement $\mu(j)$ by 1;
}
}
else if $i \notin C_k$ {
for each $j \in S_i \cap (C_k \cup C_k^{\delta})$ {
mark (i,j) for removal;
decrement $\mu(j)$ by 1;
}
for each $j \in S_i$ such that{
(i,j) is not marked for removal (noncoarse)
if (i and j share a c-point) {
mark (i,j) for removal;

decrement $\mu(j)$ by 1;
}
}
}
}

The costs are:

$$\begin{aligned} T_{2nd} &= \gamma(C_k d + (V_k - C_k)(d^2 + d)) \\ &= \gamma((V_k - C_k)d^2 + V_k d) \end{aligned} \tag{3.5}$$

Summarizing (3.2)-(3.5), we end up with

$$\begin{aligned} T_k &= \alpha(4\nu + 2log(P)) \\ &+ \beta(4 \sum_{q \in N_p} |\partial\Omega_q \cap \Omega_p| + 2log(P)) \\ &+ \gamma((V_k - C_k)d^2 + (V_k + V_{k+1} + V_{k+1}^{\delta})d + V_k + V_k^{\delta} + \sum_{q \in N_p} |\partial\Omega_q \cap \Omega_p|) \end{aligned} \tag{3.6}$$

for the k-th iteration. To estimate the overall cost of CLJP the most critical question now is how many iterations are performed and what single terms in the formula above look like.

3.3 A simple approach

To get used to certain concepts and ideas, which will then lead to a more complex model, we will now give a brief overview of an approach we used at first, which analyses a slightly simplified algorithm. The results are very poor, but some general tendencies can already be seen. When we are talking about the (local) degree d_k of point i (in the k-th iteration), in the remainder of this section, we think of it as the number of neighbors i strongly depends on that have not been categorized to be fine or coarse yet. Let us remark first, that we are allowed to express the average degree of the original graph, d_0 as an expected value:

$$E[d_0] = \sum_{i \in V_0} \frac{1}{V_0} d_0(i) \tag{3.7}$$

The same is obviously true for d_1. Denoting the difference of the degree caused by the removal of coarse and fine grid points by $d_{0,c}$ and $d_{0,f}$ yields the following relation:

$$
\begin{aligned}
E[d_1|V_1] &= \sum_{i\in V_1} \frac{1}{V_1} d_1(i) \\
&= \frac{1}{V_1} \sum_{i\in V_1} d_1(i) \\
&= \frac{1}{V_1} \sum_{i\in V_0} (d_0(i) - d_{0,c}(i) - d_{0,f}(i) \\
&= \frac{V_0}{V_1} (E[d_0] - E[d_{0,c}] - E[d_{0,f}]) \qquad (3.8)
\end{aligned}
$$

I.e. given V_1 we expressed the conditional expected value for level 1 of the algorithm in terms of expected values for level 0.

The second point to mention is, that we should expect a relation between the degree and the number of chosen coarse grid points. Every vertex $i \in V_0$ has d_0 neighbors, the set N_i; the probability of $\mu(i) > \mu(j)$ for all $j \in N_i$ is $\frac{1}{d_0+1}$. Thus, assuming constant degree, we should expect $C_0 = \frac{V_0}{d_0+1}$ and $V_1 = V_0 \frac{d_0}{d_0+1} - F_0$. As these thoughts are valid in each of the following steps too, we are now able to bound the number of steps of a CLJP-like-algorithm:

Lemma 1 *Assume, the problem is symmetric and we are using a slightly different algorithm not selecting fine grid points. Then the number of chosen coarse grid points is expected to be constant and the degree decreases by one in each step.*

Proof: If we assume, that the problem is symmetric and on average of constant degree d_k in each step, the inequalitiy

$$
\sum_{i\in V_k} d_{k,c}(i) + d_{k,f}(i) \geq 2C_k d_k \qquad (3.9)
$$

holds, as we remove at least incoming and outgoing edges for each coarse grid point. Assuming $C_k = \frac{V_k}{d_k+1}$ yields the identity:

$$
\frac{V_k - 2C_k}{V_k - C_k - F_k} = (d_k - 1) \frac{V_k}{d_k V_k - (d_k + 1)F_k} \qquad (3.10)
$$

(which is less than 1 if and only if $C_k > F_k$). The final estimation using $F_k = 0$ is thus

$$
\begin{aligned}
E[d_{k+1}|V_{k+1}] &\leq \frac{1}{V_k}(V_k d_k - 2C_k d_k) \\
&= \frac{V_k - 2C_k}{V_k - C_k - F_k} d_k \qquad (3.11) \\
&= (d_k - 1) \frac{d_k V_k}{d_k V_k - (d_k + 1)F_k} \\
&= d_k - 1 \qquad (3.12)
\end{aligned}
$$

k	V_k	C	C_k	F	F_k	d_k
0	160000	31735.5	31735.5	637.5	637.5	3.99
1	127627	58587.25	26851.75	17615	16977.5	3.01
2	83797.75	77029.75	18442.5	38186.25	20571.25	2.15
3	44784	90005.5	12975.75	54172.25	15986	1.57
4	15822.25	96178.75	6173.25	61561.5	7389.25	1.24
5	2259.75	97166.5	987.75	62699.25	1137.75	1.13
6	134.25	97228.25	61.75	62769.75	70.5	1.08
7	2	97229.25	1	62770.75	1	1

Comparison with simple model: $\sum V_k = 434427$, $\frac{1}{2}V_0 d_0 = 320000$

Table 1: Average results for the five point Laplacian, original grid.

Using this value as new estimation for the average degree d_{k+1}, we get

$$d_{k+1} = d_k - 1 = d_0 - (k+1) \tag{3.13}$$

and

$$C_{k+1} = \frac{V_{k+1}}{d_{k+1}+1} = \frac{V_k(1-\frac{1}{d_k+1})}{d_k} = \frac{V_k}{d_k+1} \tag{3.14}$$

I.e. the number of chosen coarse grid points is constant and the degree decreases by one in each expected step. □

Our algorithm is expected to use $d_0 + 1$ steps to color V_0 and we have

$$V_k = V_0 - kC_0 = V_0(1 - \frac{k}{d_0+1}) \tag{3.15}$$

and

$$\sum_{k=0}^{d_0} V_k = \frac{V_0}{d_0+1}\sum_{k=0}^{d_0} d_0 + 1 - k = \frac{V_0}{d_0+1}\sum_{k=1}^{d_0} k = \frac{1}{2}V_0 d_0 \tag{3.16}$$

We remark that the f-points play a crucial role in the algorithm: they reduce the local degree of the graph heavily, but their number is difficult to estimate, as their choice strongly depends on the geometry of the stencil. (E.g. compare 5-point and 9-point Laplacian in 2-D!) In (3.11) we should add an additional term related to the fine grid points in the numerator.

3.4 Numerical results

The following tables shows results for various 2-D problems with discrete Laplacian operators: one run with a five point stencil on a grid of 400x400 points and several runs with a nine point stencil on grids of different sizes.

k	V_k	C	C_k	F	F_k	d_k
0	97229.25	8722	8722	3355.25	3355.25	7.22
1	85152	15483.5	6761.5	16934	13578.75	4.39
2	64811.75	22432.75	6949.25	37971.25	21037.25	2.83
3	36825.25	28644.5	6211.75	55291.25	17320	1.94
4	13293.5	31804.5	3160	62642.25	7351	1.47
5	2782.5	32605.75	801.25	64256.25	1614	1.26
6	367.25	32720	114.25	64475	218.75	1.16
7	34.25	32730.25	10.25	64494.5	19.5	1.14
8	4.5	32731.5	1.25	64497.75	3.25	1.05

Comparison with simple model: $\sum V_k = 300500.25$, $\frac{1}{2}V_0 d_0 = 350997.5925$

Table 2: Average results for the five point Laplacian, first coarse grid.

k	V_k	C_k	F_k	d_k
0	28900	3157	1337.25	7.92955
1	24405.75	2363	5828.75	4.455041
2	16214	2043.25	7481.25	2.507405
3	6689.5	1246	4477.75	1.58027325
4	965.75	210	748.75	1.1913505
5	7	1.75	5.25	1

Table 3: Average results for 9 point Laplacian, 170x170 points.

k	V_k	C_k	F_k	d_k
0	32400	3785	1287.5	N/A
1	27327.5	2050.75	5362.75	
2	19914	1884.75	7107.75	
3	10921.5	1731.25	6535.75	
4	2654.5	458	1590.5	
5	606	151.5	454.5	

Table 4: Average results for 9 point Laplacian, 180x180 points.

k	V_k	C_k	F_k	d_k
0	36100	4077.5	1889.25	N/A
1	30133.25	2485	4843.75	
2	22804.5	2530	9532	
3	10742.5	1801.5	6443.25	
4	2497.75	581.5	1911.25	
5	5	1.25	3.75	

Table 5: Average results for 9 point Laplacian, 190x190 points.

k	V_k	C_k	F_k	d_k
0	90000	9960.5	3386.5	7.960044
1	76653	6264.5	12984.5	4.4253605
2	57404	6071	26314.5	2.6115815
3	25018.5	3681.75	12476	1.7304705
4	8860.75	1628.5	6231.75	1.34414075
5	1000.5	250	750.5	1.0588235

Table 6: Average results for 9 point Laplacian, 300x300 points.

3.5 Improved estimations

Given any distribution of c-points, f-points and points of a certain degree on the grid, the algorithm always performs two basic steps: it selects additional c-points and applies the heuristics, which causes the degree to be decremented locally. Therefore, we should be able to express one step of the algorithm using a mapping that describes the change of probability for any point to be of a certain degree or a certain status.

Given the cardinality of the whole set, expected values for cardinalities of subsets are derived easily.

Since the first pass, the selection of c-points, depends on the neighborhood, we introduce some additional notation: For each node $i \in V_1$ the neighborhood N_i consists of j_1 points of degree 1, ...and j_{d_0} points of degree d_0. We characterize this property by a vector $\alpha(i) = (j_1, \ldots, j_{d_0})^T \in \mathbb{N}^{d_0}$, for which necessarily $|\alpha(i)| = d(i)$ holds.

Drawing an arbitrary node i, that is neither coarse nor fine, we would like to characterize the probability, that i will become a c-point, i.e. event C occurs. The properties to have a certain neighborhood and a certain degree are described by events A_α and D_d. Note, that the events A_α, $\alpha \in N^d$ are mutually exclusive and a partition of our probability space. The equation

$$D_d = \bigcup_{|\alpha|=d} A_\alpha \tag{3.17}$$

holds and the events D_d, $d = 1, \ldots, d_0$ are a mutually exclusive partition as well.

Lemma 2 *For a node of degree d the conditional probability to be assigned as a c-point is*

$$P(C|D_d) = \sum_{|\alpha|=d} P(C|A_\alpha)P(A_\alpha|D_d). \tag{3.18}$$

The unconditional probability is then given by

$$P(C) = \sum_{d=1}^{d_0} P(D_d)P(C|D_d). \tag{3.19}$$

Proof: According to (3.17), we have

$$CD_d \quad = \quad \bigcup_{|\alpha|=d} CA_\alpha.$$

and since the events A_α are mutually exclusive, the events CA_α are too. As

$A_\alpha D_d = A_\alpha$ holds for all α, $|\alpha| = d$, we have

$$\begin{aligned} P(CD_d) &= \sum_{|\alpha|=d} P(CA_\alpha) \\ &= \sum_{|\alpha|=d} \frac{P(CA_\alpha)}{P(A_\alpha)} P(A_\alpha D_d) \end{aligned}$$

Dividing by$P(D_d) \neq 0$ finishes the proof of (3.18). This implies the second statement. □

According to the algorithm a node $i \notin C \cup F$ may be seleted to be a coarse grid point, if d is greater than or equal to the degrees of all of its neighbors; random components break ties between nodes of equal degree. Additionally, we assume, that the probability for a neighbor being of a certain degree g is independent of d. Translating this into our framework yields

$$P(C|A_\alpha) = \begin{cases} 0 & \text{if} \exists d \in \{|\alpha|+1, \ldots, d_0\}, j_d \neq 0 \\ \frac{1}{j_{|\alpha|}+1} & \text{else} \end{cases} \tag{3.20}$$

$$P(A_\alpha|D_d) = \frac{d!}{j_1! \ldots j_{d_0}!} p_1^{j_1} \ldots p_{d_0}^{j_{d_0}} \frac{1}{(1-p_c-p_f)^d} \tag{3.21}$$

and thus

$$P(C|D_d) = \frac{1}{(1-p_c-p_f)^d} \sum_{k=0}^{d} \sum_{(j_1,\ldots,j_{d-1},k,0,\ldots,0)} \frac{1}{k+1} \frac{d!}{j_1! \ldots j_{d_0}!} p_1^{j_1} \ldots p_d^k. \tag{3.22}$$

Note the scalint term in (3.21), as we are sure, that none of the neighbors is coarse or fine as well.

The heuristics change the local degree now, as certain edges are removed. Points not depending on other points any more become fine grid points.

Lemma 3 *Let every edge of $(V_k \setminus C_k, .)$ be equally likely to be removed with a probability given by Q(E)=q. Let $i \in (V_{k,d} \setminus C_{k,d})$ be an arbitrary node. Then the conditional probability to have a degree g, $0 \leq g \leq d$, after the application of the heuristics is given by*

$$P(D_g|D_d) = \binom{d}{g} q^{d-g} (1-q)^g \tag{3.23}$$

Proof: Simple combinatorics. □

3.6 Example

Let $i \in V$ be a point of degree 2 and let the maximal degree be 4. Node i will never become coarse, if one of its neighbors has a degree greater 2, i.e. if in its neighborhood $\alpha(i) = (j_1, j_2, j_3, j_4)$ the sum $j3 + j4$ is greater than zero. Only three possibilities are remaining:

$$\alpha(i) \in \{(2,0,0,0), (1,1,0,0), (0,2,0,0)\}, \tag{3.24}$$

and we have

$$P(C|D_2) = 1\frac{2!}{2!}p_1^2 + \frac{1}{2}\frac{2!}{1!1!}p_1p_2 + \frac{1}{3}\frac{2!}{2!}p_2^2 \tag{3.25}$$

3.7 Generalized notation

We now express the selection of the coarse grid and the application of the heuristics in the following framework: Let $p = (p_f, p_1, \ldots, p_{d_0}, p_c)^T \in \mathbb{R}^{d_0+2}$ be a vector describing the possibility of any point in the set to be a f-point, a points of degree 1 to d_0 or a c-point. Let P denote the set of these vectors.

Each iteration in the algorithm may then be described by a mapping

$$\begin{aligned} \Phi : P \quad &\to \quad P \qquad (3.26)\\ (p_f, p_1, \ldots, p_{d_0}, p_c)^T \quad &\mapsto (\quad p_f + \sum_{i=1}^{d_0} p_i P(\overline{C}|D_i) q^i, \\ &\qquad \sum_{i=1}^{d_0} p_i P(\overline{C}|D_i) \binom{i}{1} q^{i-1}(1-q)^1, \\ &\qquad \sum_{i=2}^{d_0} p_i P(\overline{C}|D_i) \binom{i}{2} q^{i-2}(1-q)^2, \qquad (3.27)\\ &\qquad \cdots, \\ &\qquad p_{d_0} P(\overline{C}|D_{d_0}) \binom{d_0}{d_0} q^0(1-q)^{d_0}, \\ &\qquad p_c + \sum_{i=1}^{d_0} p_i P(C|D_i))^T \end{aligned}$$

Describing the cardinalities of sets of a certain degree by expected values yields that

$$q(p) = \frac{\sum_{i=1}^{d_0} p_i P(C|D_i) i}{\sum_{i=1}^{d_0} p_i P(\overline{C}|D_i) i} \tag{3.28}$$

may be a useful estimation for Q(E). Additionally we set

$$P(C|D_i)(p) = \frac{1}{(1-p_c-p_f)^i} \sum_{j_i=0}^{i} \sum_{\alpha=(j_1,\ldots,j_i,0,\ldots,0),|\alpha|=i} \frac{1}{j_i+1} \frac{i!}{j_1!\ldots j_i!} p_1^{j_1} \ldots p_i^{j_i}. \tag{3.29}$$

according to the preceding subsection.

Then Φ is a rational function in p and we note that $P(C|D_i)(p) \leq \frac{1}{2}$, $\forall i = 1, \ldots, d_0$ is sufficient for $q(p) \leq 1$.

3.8 Some more ideas and interpretation

Note additionally, that Φ and the two passes in each iteration are closely connected to matrix operators

$$C(p) = \begin{pmatrix} 1 & & & & \\ & 1-P(C|D_1) & & & \\ & & \ddots & & \\ & & & 1-P(C|D_{d_0}) & \\ & P(C|D_1) & \ldots & P(C|D_{d_0}) & 1 \end{pmatrix}, \tag{3.30}$$

and

$$H(p, C(p)) = \begin{pmatrix} 1 & P(D_0|D_1) & \ldots & P(D_0|D_{d_0}) & 0 \\ 0 & P(D_1|D_1) & \ldots & P(D_1|D_{d_0}) & 0 \\ \vdots & & \ddots & \vdots & \vdots \\ & & & P(D_{d_0}|D_{d_0}) & 0 \\ 0 & & \ldots & 0 & 1 \end{pmatrix} \tag{3.31}$$

This fact is especially important, as the last matrix gives a hint, that we may expect a dramatic reduction in the global degree after some iterations. A closer interpretation might be useful.

It might as well be worthwile to inspect, if Φ has a linear approximation, e.g. one could try to apply a Newton-like-method to the problem to find a useful estimation for the first iterations. The model itself seems to be sufficiently precise, except for the assumptions made to determine q seem to be wrong. Instead, we should use the probability, that a neighbor has been selected as c-point and to decrease the (local) related to an expected value.

4 Computing Interpolation

The interpolation weights for $i \in F$ and $j \in C$ are determined by using the following formula:

$$\omega_{ij} = -\frac{1}{a_{ii} + \sum\limits_{k \in D_i^w} a_{ik}} (a_{ij} + \sum_{k \in D_i^s} \frac{a_{ik}\hat{a}_{kj}}{\sum\limits_{m \in C_i} \hat{a}_{km}}) \tag{4.1}$$

Here $\hat{a}$ is used as an appreviation:

$$\hat{a} = \begin{cases} a & : \quad a > 0 \\ 0 & : \quad a \leq 0 \end{cases} \tag{4.2}$$

For $i \in C$, we do not interpolate, but use the identity mapping instead.

We return to the common definition of the degree again.

4.1 The sequential algorithm

At first we are going to analyse the sequential version of the algorithm, which allows us to switch to a parallel version later by adding one single step of communication. Note that interpreting the above formula in a graph simplifies things a lot: since we forced (C1) and (H1) respectively to be fulfilled, we see that for arbitrary $i \in F$ and $j \in C$, $a_{ij} = 0$ implies $\omega_{ij} = 0$. I.e a nonzero entry ω_{ij} requires $a_{ij} \neq 0$ and we have a convenient way to find an upper bound for the number of nonzeroes in P.

We use a two pass strategy to perform all necessary computations: The first pass determines the needed amount of memory, as explained above, and initializes some things to write the matrix P in CRS format, while the second pass performs the actual computations. Both algorithms are given in pseudo code. Be aware that we neglect some minor details.

```
counter ← 0;
coarsecounter ← 0;
for each i ∈ V {
    if i ∈ C {
       coarsecounter++;
       counter++;
    }
    else {
       for each j ∈ S_i {
            if j ∈ C {
               counter++;
            }
       }
    }
}
```

Note, that we are now also able to map the indices of c-points on the fine grid to those on the coarse grid, using a mapping κ. This helps us to accessthe memory for P correctly.

The second pass finally assures that the prolongation matrix P is prepared for further use. Due to the usage of CRS format, this is done by setting up one row after another. Remark that for rows belonging to f-points, the whole row (i.e. passage in the vector p_{data}) is set to zero first (initialization phase in the first loop), before the actual computation uses a tricky distribution technique filling all slots simultaneuosly (second loop). Finally, the whole row has to be scaled (third loop).

for each $i \in V$ {
  if $i \in C$ {
    use the identity, i.e. $p_{i,\kappa(i)} \leftarrow 1$;
  }
  else{
    for each $k \in S_i$ {
      prepare entries of c-points for storing data, i.e. $p_{i,\kappa(k)} \leftarrow 0$;
      mark f-points for distribution;
    }
    $diag(i) \leftarrow a_{ii}$;
    for each $k \in N_i$ {
      if $k \in C$ {
        add a_{ik} to p_{ik}
      }
      else if $k \in D_i^s$ {
        loop over A_k, compute $\beta_k(i) = \sum_{m \in C_i} a_{km}$;
        loop over A_k, add $a_{ik}\beta_k(i)\hat{a}_{kj}$to p_{ij};

      }
      else if $k \in D_i^w$ {
        add a_{ik} to $diag(i)$
      }
    }
    for each $k \in N_i^c$ {
      scale p_{ik} by $diag(i)$;
    }
  }
}

The cost for computing interpolation thus consists of the following parts

$$\begin{aligned} T_{interpol,1st} &= \gamma\{C + (N-C)(d_s + d_c)\} \\ T_{interpol,2nd} &= \gamma\{C + (N-C)(d + 1 + d_c + d_s d + d_w + d_c)\}, \end{aligned}$$

which yields overall cost of

$$\begin{aligned} T_{interpol,seq} &= \gamma\{(N-C)(dd_s+2d+2d_c)+N+C\} && (4.3)\\ &= \gamma O(Nd^2). && (4.4) \end{aligned}$$

4.2 Implementation on a parallel machine

All we assumed so far to compute a row P_i for any $i \in V$ was, that we had access to the information stored in A_k for each $k \in N_i$ and in A_i itself. Formula (4) shows, that this information is sufficient. Going to a parallel environment, we have to make sure, that the complete rows are given for all $k \in \overline{\Omega}_p$. Thus we are able to develop a parallel version of the algorithm, after sending row information for other processors ghost points and receiving the symmetric data. In the actual implementation, this set is determined by checking required rows for computation of A^2. In this model, we neglect the cost for doing so. These thoughts yield:

$$\begin{aligned} T_{interpol,com} &= \alpha\nu + \beta d \sum_q |\partial\Omega_q \cap \Omega_p| && (4.5)\\ &= \alpha\nu + \beta O(d|\partial\Omega_p|). && (4.6) \end{aligned}$$

Summarizing (4.3) and (4.5), we have:

$$T_{interpol} = \alpha\nu + \beta O(d|\partial\Omega_p|) + \gamma O(Nd^2). \quad (4.7)$$

5 Computing RAP

In the preceding two steps we determined a coarse grid and computed an interpolation (prolongation) operator P. As the fine grid operator is known, we are now able to construct a coarse grid operator, using the Galerkin coarse grid approximation:

$$A_c = RAP \tag{5.1}$$

Using this notation, we assume $R = P^T$ (possibly in the sense of some inner product space).

If we denote $A_c = (\alpha_{ij})_{i,j \in C}$, each component of (5.1) algebraically looks like:

$$\alpha_{ij} = \sum_{k \in \Omega} r_{ik} \sum_{m \in N_k} a_{km} p_{mj} \tag{5.2}$$

which requires transposing and scaling P beforehand, of course.

5.1 The Galerkin operator

If A is symmetric and P has full rank, the Galerkin operator is an orthogonal projector respects to the A-energy inner product, and has the important property, that

$$||(I - PA_c^{-1}RA)e||_A = \min_{\tilde{e}} ||e - P\tilde{e}||_A, \, \forall e \tag{5.3}$$

holds. This is called the variational property of the Galerkin operator. Note that the degree of the Galerkin operator A_c is usually larger than the degree of A, as can be seen e.g. for the 2-D five point Laplacian easily.

5.2 The sequential algorithm

The sequential computation of RAP used in BoomerAMG is a CRS optimized matrix multiplication, based on graph algorithms. Similar to the algorithm to set up P in the previous section a two pass strategy is applied; the first pass is used to determine the number of nonzeroes and to set up temporary vectors, the second one to compute the product row-by-row. We give a brief sketch of the idea of the algorithm:

According to (5.2) the problem should be encountered from a graph theoretical point of view, the algorithm itself is just a slight variation of a depth first search. To figure out, where nonzero entries in the i-th row of the triple matrix product RAP occur, we have to check all columns k in R_i with nonzero entries. These are related to rows A_k with columns m, which are related to rows P_m with columns j again. Once a nonzero entry p_{mj} has been found, a new nonzero α_{ij} is discovered. After all edges p_{mj} were visited, P_m has not to be taken into account again for any other k. We are allowed to mark the m-th row of P (is associated with the m-th column of A) as treated. As the entry α_{ij} might be discovered from a different node $\tilde{m}$ in the future again, we should mark j itself

as being discovered in a search starting at i too. Setting these flags requires two temporary vectors, which are both initialized with -1, and set to i in the m-th slot, whenever a row m has been treated or respectively to the number of points already discovered.

If A is symmetric, the Galerkin operator is symmetric, too, and we rewrite (5.2) as

$$\alpha_{ij} = \alpha_{ji} = (\varepsilon_j, P^T A P \varepsilon_i), \tag{5.4}$$

i.e. we are allowed two use a simple stencil multiplication scheme to determine the necessary number of multiplications. For each c-point, we have to visit all its neighbors, apply the fine grid operator to each of them and finally restrict the values for all points affected by this operation back to neighboring c-points. This yields the estimation

$$T_{RAP,seq} = O(Nd^2) \tag{5.5}$$

Recognize that the less c-points are selected, the faster the construction of the coarse grid operator will be. Indeed, tests show enormous differences between the seven point Laplacian and an articial 27-point-stencil.

5.3 The parallel variant

As in the previous sections, the problem of computation is inherently local again, though we have to investigate carefully, which communication steps are still required. If we store all information P_i, $i \in \overline{\Omega}_p$, on processor p, we are able to compute the contributionof processor p to A_c, denoted by $A_c^{(p)}$. This represents the operation of the original operator A on the points in Ω_p. Since the sets Ω_q are a partition of the original domain Ω, we have

$$\begin{aligned} \alpha_{ij} &= \sum_{p\in\Pi} \sum_{k\in\Omega_p} r_{ik} \sum_{m\in N_k} a_{km} p_{mj} && (5.6)\\ &= \sum_{p\in\Pi} \alpha_{ij}^{(p)}. && (5.7) \end{aligned}$$

Thus, compared to the sequential version, two communication additional phases are required: we have to submit/receive one layer of boundary data for P, compute and distribute the contributions to neighboring processes.

send p_m, $m \in \partial\Omega_q$;
receive p_m, $m \in \partial\Omega_p$;
compute $\alpha_i^{(p)}$, $i \notin (\Omega_p \cap C)$;
send $\alpha_i^{(p)}$, $i \notin (\Omega_p \cap C)$;
receive $\alpha_i^{(q)}$, $i \in (\Omega_p \cap C)$;
add $\alpha_i^{(q)}$, $i \in (\Omega_p \cap C)$;

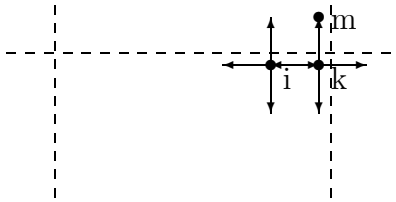

Figure 1: Example

We remark, that we already took into account the additional cost for computation, as for a c-point close to the boundary (5.5) already recognizes restriction from non-local f-points. The time needed for a parallel computation of the coarse grid operator is thus

$$T_{op,par} = T_{op,seq} + T_{op,comm}, \tag{5.8}$$

with

$$T_{op,comm} = 2\alpha\nu + \beta(\sum_{q \in N_p} |\Omega_p \cap \partial\Omega_q| d + |\partial\Omega_p \cap C| d_c) \tag{5.9}$$

which yields

$$T_{op,par} = \alpha\nu + \beta O(|\partial\Omega_p| d) + \gamma O(n d^2) \tag{5.10}$$

6 Test results

In the following subsection, we show representative results of the performed tests, stressing the differences between the behavior in sequential and parallel runs. **Scenario one** refers to sequential runs, performed on a linux box, with two Intel Xeon processors and 512 MB RAM; **scenario two** indicates runs on IBM ASCI Blue Pacific, using a topology of 4x4x4 processors, i.e. on 16 of 272 nodes, each with 1.5 GB memory. Although the architectures are different, general tendencies can be discovered. The following two tables show results for different numbers of gridpoints.

Figure 3 shows an enormous amount of time used for constructing the interpolation operator, although the measured effort for communication is rather small. As the latter fact matches our theoretical results, we have to conclude that aspects not described by the model play a crucial role. (Although these thoughts are somewhat hypothetical, a closer look at the actual implementation shows, that one part of the innermost loop of the algorithm described on p. 21, the computation of $\beta_k(i)$, requires a mapping of global indices to local indices for off-processor points. These searches are performed in $O(log|\partial\Omega_p|))$, a considerably small value, but still may cause the factor close to 4 we see, when we compare the parallel and sequential values relative to cost of interpolation in the first step.) Additionally note the differing behavior of coarsening, which partly seems to be related to CLJP, as it already occurs on level 1 and 2.

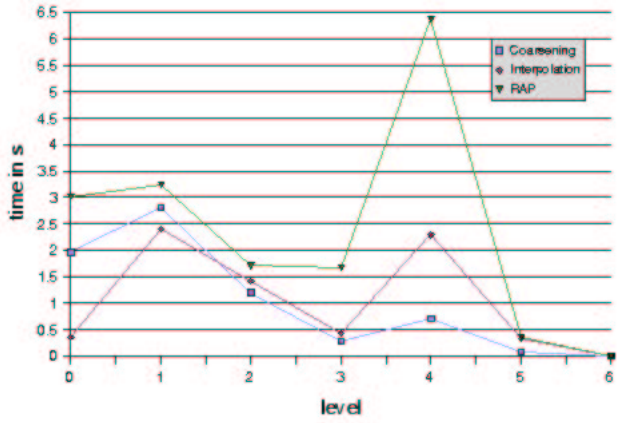

Figure 2: Scenario 1, 92x92x92 points.

.

l	$T_{coarsen}$	$T_{interpol}$	T_{RAP}	T_{total}	n	nz	d_{min}	d_{max}	d_{ave}
0	1.96 (36.8%)	0.35(0.07%)	3.01(56.60%)	5.32	778688	5400032	4	7	6.9
1	2.81 (33.3%)	2.4 (28.4%)	3.24 (38.3%)	8.45	389344	7245736	7	19	18.6
2	1.2 (27.7%)	1.42 (32.8%)	1.71 (39.5%)	4.33	69017	2364793	14	40	34.3
3	0.27 (11.4%)	0.43 (18.1%)	1.67 (70.5%)	2.37	9820	587748	26	81	59.9
4	0.7 (7.5%)	2.29 (24.4%)	6.38 (68.1%)	9.37	2813	667147	95	334	237.2
5	0.07 (9.2%)	0.33 (43.4%)	0.36 (47.4%)	0.76	362	96882	169	349	267.6
6	0	0	0	0	31	961	31	31	31
7	0	0	0	0	2	4	2	2	2

Setup: 30.81 s, equivalent to 38.4 V-Cycles;
Solver:4.81, 6 V-Cycles.

Table 7: 7pt Laplacian, (nx, ny, nz) = (92, 92, 92), (Px, Py, Pz) = (1, 1, 1)

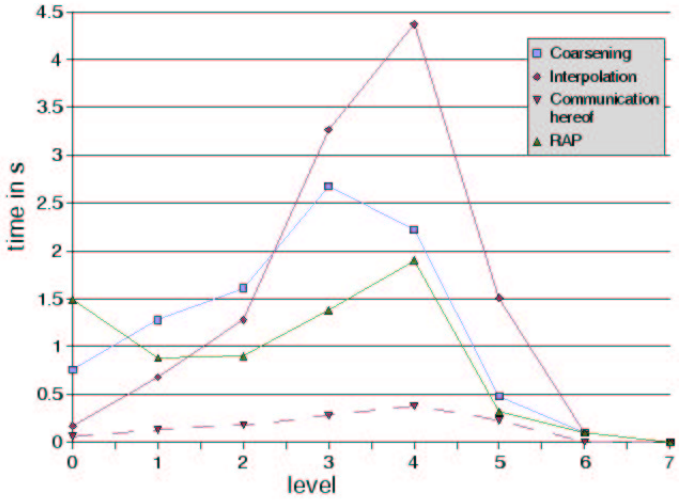

Figure 3: Scenario 2, 128x128x128 points.

.

l	$T_{coarsen}$	$T_{interpol}$	T_{RAP}	T_{total}	n	nz	d_{min}	d_{max}	d_{ave}
0	0.76 (31.4%)	0.17 (7.0%)	1.49 (61.6%)	2.42	2097152	14581760	4	7	7
1	1.28 (45.0%)	0.68 (23.9%)	0.88 (31.0%)	2.84	1049989	19638565	7	19	18.7
2	1.61 (42.5%)	1.28 (33.8%)	0.90 (23.8%)	3.79	206294	8409110	12	73	40.8
3	2.68 (36.6%)	3.27 (44.6%)	1.38 (18.8%)	7.33	36854	4201462	28	173	114
4	2.22 (26.2%)	4.37 (51.5%)	1.90 (22.4%)	8.49	4951	1233909	65	379	249.2
5	0.48 (20.8%)	1.51 (65.4%)	0.32 (13.9%)	2.31	633	206143	169	574	325.7
6	0.01 (33.3%)	0.01 (33.3%)	0.01 (33.3%)	0.03	36	1296	36	36	36
7	0	0	0	0	10	100	10	10	10

Setup: 27.19 seconds, equivalent to 31.43 V-cycles;
Solver: 6.92 seconds, 8 V-cycles;

Table 8: 7pt Laplacian, (nx, ny, nz) = (128, 128, 128), (Px, Py, Pz) = (4, 4, 4)

References

[1] V.E. Henson and U. Meier Yang, *BoomerAMG: a Parallel Algebraic Multigrid Solver and Preconditioner*, Applied Numerical Mathematics 41 (2002), 155-177.

[2] K. Gallivan and U. Meier Yang, *Efficiency Issues in Parallel Coarsening Schemes*, to be published.

[3] M.T. Jones and P.E. Plassmann, *A Parallel Graph Coloring Heuristic*, SIAM Journal of Scientific Computing 14 (1993) 654-669

[4] A.J. Cleary, R.D. Falgout, V.E. Henson, and J.E. Jones, *Coarse-grid Selection for Parallel Algebraic Multigrid*, Technical report

[5] R. Barrett, et al., *Templates for the solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM (1994)